STICKY Z BIT

BACKGROUND OF THE INVENTION

Field of the Invention:

The present invention relates to systems and methods for indicating the status of an ALU operation and, more particularly, to systems and methods for indicating the status of an ALU mathematical operation, pursuant to which the result is based on one or more prior results.

Description of Prior Art:

Processors, including microprocessors, digital signal processors and microcontrollers, operate by running software programs that are embodied in one or more series of instructions stored in a memory. The processors run the software by fetching the instructions from the series of instructions, decoding the instructions and executing them. A 16-bit arithmetic logic unit (ALU) of a processor, including digital signal processors, are conventionally adept at processing ALU mathematical operation instructions, such as addition operation instructions, that operate on a data word or data byte, and indicates the various statuses affected by the execution of each ALU mathematical operation instruction with status flags. For example, a 16-bit ALU of a processor is adept at performing an addition operation according to an addition mathematical operation instruction on operands represented as a data word and indicating if the result of the addition operation produced an arithmetic result of zero by setting a zero status bit. In general, the statuses affected by

a 16-bit ALU mathematical operation is set based on each single 16-bit result produced by the ALU mathematical operation. Accordingly, the statuses affected by a 16-bit ALU mathematical operation may be misleading in conditions where the result is based on one or more prior results and produced by executing a series of ALU mathematical operation instructions. These type of ALU mathematical operations execute in two processing cycles and produce a 16-bit results in each of the processing cycles to form the final result. As a result, the status flags may only indicate the statuses affected by the ALU mathematical operation based on a result produced during the most recent processing cycle instead of the results produced during both processing cycles.

There is a need for a new method of indicating the statuses affected by the performance of an ALU mathematical operation. There is a further need for a new method of indicating the statuses affected by the performance of an ALU mathematical operation producing a final result formed by two distinct semi-results. There is a further need for a new method of indicating the statuses affected by the performance of an ALU mathematical operation for each result produced during the performance of the ALU mathematical operation based on preceding results.

SUMMARY OF THE INVENTION

According to embodiments of the present invention, a method and a processor for indicating a status affected by the performance of an ALU mathematical operation are provided. The status affected by the performance of an ALU mathematical operation

include the sticky zero flag for indicating whether a result of the ALU mathematical operation produced an arithmetic result of zero. The pre-requisite for employing the sticky zero flag is that a prior math operation must be a conventional math operation (i.e., no carry involved) that set the sticky zero flag with a one (1) or a zero (0) and the subsequent math operation is a math operation including a carry. The sticky zero flag maintains a setting indicating non-zero status upon the production of a non-zero result until the performance of a ALU mathematical operation not including a carry produces an arithmetic result of zero.

A method of indicating a status affected by the performance of an ALU mathematical operation according to an embodiment of the present invention includes executing an ALU mathematical operation instruction on a set of source operands. The method further includes determining that the ALU mathematical operation instruction corresponds to an ALU mathematical operation instruction with carry. The method further includes producing a result based on the set of source operands in accordance with the ALU mathematical operation instruction and setting a status flag based on the result.

In an embodiment of the present invention, the setting of the status flag includes determining that the result is a non-zero value. Upon determining that the result is a non-zero value the status flag is cleared by writing to it a value of zero. The value of zero is maintained in the status flag until an ALU mathematical operation instruction without carry is determined.

In an embodiment of the present invention, the setting of the status flag includes determining that the result is a zero value. Upon determining that the result is a zero, the value of the status flag is maintained.

A processor for indicating a status affected by the performance of an ALU mathematical operation includes an ALU operable to execute an ALU mathematical operation instruction on a set of source operands. The ALU can determines that the ALU mathematical operation instruction corresponds to an ALU mathematical operation instruction with carry. The ALU is further operable to produce a result based on the set of source operands in accordance with the ALU mathematical operation instruction and set a status flag based on the result.

BRIEF DESCRIPTION OF THE DRAWINGS

The above described features and advantages of the present invention will be more fully appreciated with reference to the detailed description and appended figures in which:

Fig. 1 depicts a functional block diagram of an embodiment of a processor chip within which embodiments of the present invention may find application;

Fig. 2 depicts a functional block diagram of a data busing scheme for use in a processor, which has a microcontroller and a digital signal processing engine, within which embodiments of the present invention may find application;

Fig. 3 depicts a functional block diagram of a processor configuration for indicating a status affected by the performance of an ALU mathematical operation according to embodiments of the present invention; and

Fig. 4 depicts a method of indicating a status affected by the performance of an ALU mathematical operation according to embodiments of the present invention

DETAILED DESCRIPTION OF THE INVENTION

According to embodiments of the present invention, a method and a processor for indicating a status affected by the performance of an ALU mathematical operation are provided. The status affected by the performance of an ALU mathematical operation include the sticky zero flag for indicating whether a result of the ALU mathematical operation produced an arithmetic result of zero. Moreover, the sticky zero flag indicates whether a result produced during the performance of the ALU mathematical operation produced an arithmetic result of zero. The sticky zero flag maintains a setting indicating non-zero status until a subsequent ALU mathematical math operation without carry produces a result of zero.

In order to describe embodiments of ALU mathematical operation status indicating, an overview of pertinent processor elements is first presented with reference to Figs. 1 and 2. The ALU mathematical operation status indicating is then described more particularly with reference to Figs. 3-5.

Overview of Processor Elements

Fig. 1 depicts a functional block diagram of an embodiment of a processor chip within which the present invention may find application. Referring to Fig. 1, a processor 100 is coupled to external devices/systems 140. The processor 100 may be any type of

processor including, for example, a digital signal processor (DSP), a microprocessor, a microcontroller or combinations thereof. The external devices 140 may be any type of systems or devices including input/output devices such as keyboards, displays, speakers, microphones, memory, or other systems which may or may not include processors. Moreover, the processor 100 and the external devices 140 may together comprise a stand alone system.

The processor 100 includes a program memory 105, an instruction fetch/decode unit 110, instruction execution units 115, data memory and registers 120, peripherals 125, data I/O 130, and a program counter and loop control unit 135. The bus 150, which may include one or more common buses, communicates data between the units as shown.

The program memory 105 stores software embodied in program instructions for execution by the processor 100. The program memory 105 may comprise any type of nonvolatile memory such as a read only memory (ROM), a programmable read only memory (PROM), an electrically programmable or an electrically programmable and erasable read only memory (EPROM or EEPROM) or flash memory. In addition, the program memory 105 may be supplemented with external nonvolatile memory 145 as shown to increase the complexity of software available to the processor 100. Alternatively, the program memory may be volatile memory which receives program instructions from, for example, an external non-volatile memory 145. When the program memory 105 is nonvolatile memory, the program memory may be programmed at the time of manufacturing the processor 100 or prior to or during implementation of the processor

100 within a system. In the latter scenario, the processor 100 may be programmed through a process called in-line serial programming.

The instruction fetch/decode unit 110 is coupled to the program memory 105, the instruction execution units 115 and the data memory 120. Coupled to the program memory 105 and the bus 150 is the program counter and loop control unit 135. The instruction fetch/decode unit 110 fetches the instructions from the program memory 105 specified by the address value contained in the program counter 135. The instruction fetch/decode unit 110 then decodes the fetched instructions and sends the decoded instructions to the appropriate execution unit 115. The instruction fetch/decode unit 110 may also send operand information including addresses of data to the data memory 120 and to functional elements that access the registers.

The program counter and loop control unit 135 includes a program counter register (not shown) which stores an address of the next instruction to be fetched. During normal instruction processing, the program counter register may be incremented to cause sequential instructions to be fetched. Alternatively, the program counter value may be altered by loading a new value into it via the bus 150. The new value may be derived based on decoding and executing a flow control instruction such as, for example, a branch instruction. In addition, the loop control portion of the program counter and loop control unit 135 may be used to provide repeat instruction processing and repeat loop control as further described below.

The instruction execution units 115 receive the decoded instructions from the instruction fetch/decode unit 110 and thereafter execute the decoded instructions. As part

of this process, the execution units may retrieve one or two operands via the bus 150 and store the result into a register or memory location within the data memory 120. The execution units may include an arithmetic logic unit (ALU) such as those typically found in a microcontroller. The execution units may also include a digital signal processing engine, a floating point processor, an integer processor or any other convenient execution unit. A preferred embodiment of the execution units and their interaction with the bus 150, which may include one or more buses, is presented in more detail below with reference to Fig. 2.

The data memory and registers 120 are volatile memory and are used to store data used and generated by the execution units. The data memory 120 and program memory 105 are preferably separate memories for storing data and program instructions respectively. This format is a known generally as a Harvard architecture. It is noted, however, that according to the present invention, the architecture may be a Von-Neuman architecture or a modified Harvard architecture which permits the use of some program space for data space. A dotted line is shown, for example, connecting the program memory 105 to the bus 150. This path may include logic for aligning data reads from program space such as, for example, during table reads from program space to data memory 120.

Referring again to Fig. 1, a plurality of peripherals 125 on the processor may be coupled to the bus 150. The peripherals may include, for example, analog to digital converters, timers, bus interfaces and protocols such as, for example, the controller area

network (CAN) protocol or the Universal Serial Bus (USB) protocol and other peripherals. The peripherals exchange data over the bus 150 with the other units.

The data I/O unit 130 may include transceivers and other logic for interfacing with the external devices/systems 140. The data I/O unit 130 may further include functionality to permit in circuit serial programming of the Program memory through the data I/O unit 130.

Fig. 2 depicts a functional block diagram of a data busing scheme for use in a processor 100, such as that shown in Fig. 1, which has an integrated microcontroller arithmetic logic unit (ALU) 270 and a digital signal processing (DSP) engine 230. This configuration may be used to integrate DSP functionality to an existing microcontroller core. Referring to Fig. 2, the data memory 120 of Fig. 1 is implemented as two separate memories: an X-memory 210 and a Y-memory 220, each being respectively addressable by an X-address generator 250 and a Y-address generator 260. The X-address generator may also permit addressing the Y-memory space thus making the data space appear like a single contiguous memory space when addressed from the X address generator. The bus 150 may be implemented as two buses, one for each of the X and Y memory, to permit simultaneous fetching of data from the X and Y memories.

The W registers 240 are general purpose address and/or data registers. The DSP engine 230 is coupled to both the X and Y memory buses and to the W registers 240. The DSP engine 230 may simultaneously fetch data from each the X and Y memory, execute instructions which operate on the simultaneously fetched data and write the result to an

accumulator (not shown) and write a prior result to X or Y memory or to the W registers 240 within a single processor cycle.

In one embodiment, the ALU 270 may be coupled only to the X memory bus and may only fetch data from the X bus. However, the X and Y memories 210 and 220 may be addressed as a single memory space by the X address generator in order to make the data memory segregation transparent to the ALU 270. The memory locations within the X and Y memories may be addressed by values stored in the W registers 240.

Any processor clocking scheme may be implemented for fetching and executing instructions. A specific example follows, however, to illustrate an embodiment of the present invention. Each instruction cycle is comprised of four Q clock cycles Q1 – Q4. The four phase Q cycles provide timing signals to coordinate the decode, read, process data and write data portions of each instruction cycle.

According to one embodiment of the processor 100, the processor 100 concurrently performs two operations – it fetches the next instruction and executes the present instruction. Accordingly, the two processes occur simultaneously. The following sequence of events may comprise, for example, the fetch instruction cycle:

Q1: Fetch Instruction
Q2: Fetch Instruction
Q3: Fetch Instruction
Q4: Latch Instruction into prefetch register, Increment PC

The following sequence of events may comprise, for example, the execute instruction cycle for a single operand instruction:

Q1: latch instruction into IR, decode and determine addresses of operand data
Q2: fetch operand
Q3: execute function specified by instruction and calculate destination address for data
Q4: write result to destination

The following sequence of events may comprise, for example, the execute instruction cycle for a dual operand instruction using a data pre-fetch mechanism. These instructions pre-fetch the dual operands simultaneously from the X and Y data memories and store them into registers specified in the instruction. They simultaneously allow instruction execution on the operands fetched during the previous cycle.

Q1: latch instruction into IR, decode and determine addresses of operand data
Q2: pre-fetch operands into specified registers, execute operation in instruction
Q3: execute operation in instruction, calculate destination address for data
Q4: complete execution, write result to destination

ALU Mathematical Operation Status Indicating

Fig. 3 depicts a functional block diagram of a processor for indicating a status of an ALU mathematical operation according to the present invention. Referring to Fig. 3, the processor includes a program memory 300 for storing instructions such as ALU mathematical operation instructions. The processor also includes a program counter 305 which stores a pointer to the next program instruction that is to be fetched. The processor further includes an instruction register 315 for storing an instruction for execution that has

been fetched from the program memory 300. The processor may further include pre-fetch registers (not shown) that may be used for fetching and storing a series of upcoming instructions for decoding and execution. The processor also includes an instruction decoder 320, an arithmetic logic unit (ALU) 325, registers 345 and a status register 350.

The instruction decoder 320 decodes instructions, such as ALU mathematical operation instructions, that are stored in the instruction register 315. Based on the combination of bits in the instruction, the instruction decoder 320 decodes particular bits in ALU mathematical operation instructions that results in the selective activation of logic within the ALU 325 for fetching operands, performing the operation specified by the fetched instruction on the operands, producing an output/result in accordance with the instruction to the appropriate data memory location and setting a status bit in accordance with the operation performed on the operands by the instruction.

The ALU 325 includes registers 330 that may receive one or more operands from the registers 345 and/or a data memory location 355. The origin of the one or more operands depends on the addressing mode defined by the combination of bits used in the instruction. For example, one combination of address mode bits results in the activation of logic that obtains one operand from data memory across the X data bus depicted in Fig. 2, another operand from a register.

The ALU 325 includes ALU logic 335 which may receive the one or more operands from the registers 330. The ALU logic 335 executes arithmetic and logic operations according to instructions, such as ALU mathematical operation instructions, decoded by the instruction decoder on the one or more operands fetched from the registers

345 and/or from address location in the data memory 345. The ALU logic 335 produces outputs/results in accordance with the arithmetic and logic operations based on the one or more operands to one of registers 345 and/or the status register 350. The outputs/results may be stored/written to the register in accordance with bits specified in the instruction.

Status register 350 contains status bits that indicate the status of processor elements and operations. Status register 350 may be a 16-bit status register. The status register 350 may be separated into a lower segment and an upper segment. The processor operations for which status may be indicated include MCU ALU, DSP Adder/Subtractor, repeat, and Do loop. A bit in the status register 350, such as a sticky z bit, can indicate whether an operation, such as an ALU mathematical operation, produce an arithmetic result of zero. An ALU mathematical operation may comprise of a series of ALU mathematical operation instructions, some specifying a carry and some not specifying a carry, but each producing an arithmetic result. The bit is set, such as with a value of one (1) or zero (0), to indicate that an ALU mathematical operation instruction has been executed some time in the past which produced a zero arithmetic result or a non-zero arithmetic result. The bit is initially set with a value of one (1) by the execution of an ALU math operation instruction without carry produces a zero result. The bit remains set until a subsequent ALU mathematical operation instruction with carry is executed and produces a non-zero arithmetic result. When a non-zero result is produced, the bit is cleared, such as with a value of zero, and remains cleared until a subsequent ALU mathematical operation instruction without carry is executed and produces a zero arithmetic result. The application of the bit, such as the sticky z bit, is determined by the execution of an ALU mathematical operation instruction

with carry subsequent to the execution of an ALU mathematical operation instruction without a carry.

Fig. 4 depicts a method of indicating a status of an ALU mathematical operation according to the present invention. Fig. 4 is best understood when viewed in conjunction with Fig. 3. Referring to Fig. 4, in step 400, the decoder 320 determines whether an ALU mathematical operation instruction specifies a carry operation. The instruction may be fetched from the program memory 300. The decoder determines whether the ALU mathematical operation instruction specifies a carry operation by decoding bits in the instruction. If an ALU mathematical operation instruction with carry is determined, then the method proceeds to step 405. In step 405, the ALU mathematical operation instruction is executed by ALU 325. The ALU performs an operation on one or more operands specified by the ALU mathematical operation instruction. Then in step 410, the ALU 325 produces an output/result in accordance with the performed operation. The output result may be stored in data memory or data registers as specified by the ALU mathematical operation instruction. In step 415, the ALU 325 determines whether the output/result produced an arithmetic result of zero. If the ALU did not produce an arithmetic result of zero, then the method proceeds to step 420 where the status bit is cleared, that is changing the bit value from one (1) to zero (0). The status bit maintains the value of zero until an ALU mathematical operation instruction is executed that is not an ALU mathematical operation instruction specifying a carry operation. If the ALU produced an arithmetic result of zero, then the method proceeds to step 425. In step 425, the ALU 325 does

nothing until the next ALU mathematical operation instruction not specifying a carry is executed and produces a zero arithmetic result.

If an ALU mathematical operation instruction with carry is not determined, then the method proceeds to step 435. In step 435, the ALU mathematical operation instruction is executed by ALU 325. The ALU performs an operation on one or more operands specified by the ALU mathematical operation instruction. Then in step 440, the ALU 325 produces an output/result in accordance with the performed operation. The output result may be stored in data memory or data registers as specified by the ALU mathematical operation instruction. In step 445, the ALU 325 determines whether the output/result produced an arithmetic result of zero. If the ALU did not produce an arithmetic result of zero, then the method proceeds to step 455 where the status bit is set with a value of zero. If the ALU produced an arithmetic result of zero, then the method proceeds to step 450. In step 450, the status bit is set with a value of 1.

While specific embodiments of the present invention have been illustrated and described, it will be understood by those having ordinary skill in the art that changes may be made to those embodiments without departing from the spirit and scope of the invention.